LAPORAN TUGAS KECIL III

IF2211 STRATEGI ALGORITMA

IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Disusun oleh :

Go Dillon Audris (13521062)

Bill Clinton (13521064)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022/2023

DAFTAR ISI

Bab 1 : Pendahuluan

Algoritma *Uniform Cost Search* (UCS) dan *A star* (A*) adalah dua algoritma yang dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik yang lain. Pada tugas kecil III IF2211 Strategi Algoritma ini, kami akan menentukan lintasan terpendek dari jalan-jalan di kota Bandung berdasarkan peta Google Map. Ruas-ruas jalan tersebut akan direpresentasikan sebagai graf. Simpulnya menyatakan persilangan jalan (simpang 3, 4, atau 5) atau ujung jalan dan bobot grafnya menyatakan jarak antar simpul. Jalannya diasumsikan dapat dilalui dari dua arah. Untuk menghitung jarak antara dua simpul, dapat digunakan rumus jarak Euclidean (berdasarkan koordinat) atau digunakan ruler di Google Map atau cara lain yang tersedia pada Google Map.

Pada program yang kami buat, kami membuat graf yang merepresentasikan peta pada area tertentu. Berdasarkan graf tersebut, program akan menerima input simpul asal dan simpul tujuan. Setelah itu, program akan menentukan lintasan terpendek antara kedua simpul menggunakan algoritma UCS dan A*. Nilai heuristik yang dipakai dalam pembuatan program ini adalah garis lurus dari suatu titik ke tujuan.

Bab 2 : Algoritma untuk Menentukan Lintasan Terpendek

2.1 Algoritma Uniform Cost Search (UCS)

Algoritma *Uniform Cost Search* (UCS) adalah salah satu jenis algoritma *uninformed search* yang dapat digunakan untuk menentukan lintasan terpendek dari suatu simpul ke simpul yang lain. Algoritma UCS menjamin ditemukannya solusi optimal. Dalam algoritma UCS, dari simpul awal, akan dikunjungi simpul-simpul yang yang bersebelahan dan akan dipilih simpul yang jaraknya paling pendek berikutnya dari semua simpul yang belum dikunjungi dan yang bersebelahan dengan simpul yang telah dikunjungi hingga akhirnya ditemukan jarak terpendek dari simpul awal ke simpul tujuan. Pada program yang kami buat, digunakan struktur data *set* untuk menampung simpul-simpul yang telah dikunjungi sebelumnya untuk mencegah adanya nilai simpul yang duplikat. Selain itu, digunakan juga struktur data *priority queue* yang akan memberikan simpul dengan jarak terpendek berikutnya dari seluruh simpul yang bersebelahan dengan simpul yang telah dikunjungi sebelumnya.

Untuk pencarian lintasan terpendeknya, pertama, struktur data set yang berisi simpul yang telah dikunjungi dan *priority queue* diinisialisasi. *Priority queue* diisi terlebih dahulu dengan simpul awal (*start node*). Lalu, selama *queue* tidak kosong, dilakukan iterasi dengan mengambil simpul sekarang/*current node* (bersama *total weight* yang melambangkan total jarak dan *path* yang melambangkan lintasan menuju tujuan) dari *priority queue* dan membandingkannya dengan simpul tujuan. Jika kedua simpul tersebut tidak sama, dilanjutkan pengecekan apakah *current node* ada dalam *set* yang berisi simpul yang telah dikunjungi sebelumnya. Jika tidak ada, *current node* dimasukkan ke dalam *set* tersebut. Kemudian, dilakukan pengecekan terhadap *neighbor node* dari *current node* beserta bobot sisinya (*edge weight*). Jika *neighbor node* belum dikunjungi sebelumnya (tidak ada di dalam *set*) dan *edge weight* lebih dari 0 (menandakan ada lintasan), *path* ditambahkan dengan *current node* menjadi *path* yang baru (*new path*) dan *total weight* ditambahkan dengan *edge weight* menjadi bobot yang baru (*new distance*). *Path* dan bobot yang baru ini kemudian akan dimasukkan kembali ke *priority queue*. Iterasi akhirnya berhenti ketika *current node*-nya sama dengan simpul tujuan/*goal node*. Sebelum iterasi ini dihentikan, akan tercatat *path* dan bobot/jarak lintasan totalnya yang menandakan lintasan hasil dengan jarak terpendek dengan algoritma UCS (hal ini bisa terjadi karena

fungsi get untuk priority queue mengeluarkan nilai dari queue dengan nilai terendah, yang dalam kasus ini adalah *total weight*).

2.2 Algoritma *A Star* (A*)

Algoritma *A star* (A*) adalah salah satu jenis *informed search* yang dapat digunakan untuk untuk menentukan lintasan terpendek dari suatu simpul ke simpul yang lain. Algoritma A* juga menjamin ditemukannya solusi optimal. Algoritma A* menggabungkan optimalitas dari algoritma UCS serta kecepatan dari algoritma *Greedy Best-First-Search* untuk mendapatkan rute yang optimal dengan biaya waktu yang tidak terlalu lama.

Dalam algoritma A*, dengan simpul asal menjadi simpul ekspan pertama, akan dibangkitkan simpul-simpul yang bersebelahan dengan simpul ekspan. Simpul ekspan berikutnya akan dipilih dari simpul-simpul hidup yang belum dikunjungi dan memiliki perkiraan evaluasi biaya yang terpendek. Perkiraan evaluasi ini dihitung dengan menjumlahkan biaya atau jarak dari simpul asal ke simpul hidup dengan nilai heuristik dari simpul hidup. Nilai heuristik yang dipakai dalam algoritma A* haruslah *admissible*, yaitu nilai tidak boleh *overestimate* atau melebihi jarak atau biaya sebenarnya. Pencarian rute akan terus dilakukan atau diiterasi hingga simpul ekspan merupakan simpul tujuan yang dicari. Di akhir pencarian, akan didapatkan rute terpendek dari simpul asal ke simpul tujuan beserta nilainya.

Pada program yang kami buat, digunakan struktur data *set* untuk menampung simpul-simpul ekspan yang telah dikunjungi sebelumnya untuk mencegah adanya kasus simpul dikunjungi lebih dari satu kali. Selain itu, digunakan juga struktur data *priority queue* yang akan memberikan simpul dengan perkiraan evaluasi biaya terpendek berikutnya dari seluruh simpul yang sedang hidup. Nilai heuristik tiap simpul diperoleh dengan memanfaatkan matriks nilai heuristik yang diperoleh dari hasil *parsing file*.

Dalam implementasinya, algoritma A* dilakukan dengan pertama-tama membangkitkan simpul asal dengan cara memasukkan sebuah *tuple* berisi simpul asal, *list* rute kosong, dan perkiraan evaluasi biaya simpul asal. Selanjutnya, selama pencarian belum selesai, maka akan dilakukan iterasi dengan langkah sebagai berikut:

1. Mengambil elemen pertama dalam *priority queue.* Elemen pertama adalah elemen dengan perkiraan evaluasi biaya terpendek / terkecil.
2. Jika simpul ekspan dari elemen yang diambil bukanlah simpul tujuan, maka bangkitkan seluruh simpul yang bertetangga dengan simpul ekspan yang belum

dikunjungi dan masukkan *tuple* berisi simpul tersebut, *list* rute sejauh ini, dan perkiraan evaluasi biaya simpul tersebut (biaya dari simpul asal ke simpul hidup ditambah nilai heuristik). Masukkan simpul ekspan sebelumnya ke dalam set untuk menandai bahwa simpul telah dikunjungi. Kembali ke langkah 1.

3. Jika simpul ekspan dari elemen yang diambil adalah simpul tujuan, maka pencarian selesai.

Bab 3 : *Source Code*

3.1 *File* AStar.py

```python
# File : AStar.py
# Contains the class AStar, that are used for
# Finding the shortest path on a graph using the A* algorithm

from queue import PriorityQueue

class AStar:

    # CONSTRUCTOR
    # Construct and initialize the object
    def __init__(self, nodeName, map, heuristicMap):
        self.nodeName = nodeName                    # list of name of node, used for showing the true path
        self.map = map                              # adjacency matrix used to represent the graph
        self.heuristicMap = heuristicMap            # matrix of heuristic value, store the straight line distance of node of row i to node of column j
        self.queue = PriorityQueue()                # queue used to process and pick the node to expand
        self.visited = set()                        # set to store all nodes that are already visited
        self.found = False                          # boolean flag to know is the result path is valid or not
        self.resultPath = []                        # list of the node that create the shortest path from start node to goal node
        self.distance = 0                           # the optimum distance from start node to goal node

    # GETTER
    # Get the result path
    def getResultPath(self):
        return self.resultPath

    # Get the distance
    def getDistance(self):
        return self.distance

    # Get the validity of the result path
    def getFound(self):
        return self.found

    # HELPER METHOD
    # Return string that contains all of node name from the result path
    def getPathName(self):
        pathName = ""

        for i in range(0, len(self.resultPath)-1):
            pathName += self.nodeName[self.resultPath[i]] + " - "

        pathName += self.nodeName[self.resultPath[len(self.resultPath)-1]]

        return pathName
```

```python
    # A* ALGORITHM
    # Awaken the node adjacent to the expand node and put it to the queue
    def awakenNode(self, tempCost, tempNode, tempPath, goalNode):
        # Put the expand node to visited set
        self.visited.add(tempNode)

        # Put all node adjacent to expand node and not visited to the queue
        for i in range(0, len(self.map)):
            if (i not in self.visited and self.map[tempNode][i] != 0):
                self.queue.put((tempCost + self.map[tempNode][i] + self.heuristicMap[i][goalNode], i, tempPath + [tempNode]))

    # Start the A* algorithm until the result path from start node to goal node is found
    def findPath(self, startNode, goalNode):
        # Awaken the start node
        self.queue.put((0 + self.heuristicMap[startNode][goalNode], startNode, []))

        # Looping until the goalNode is found
        while (not self.found and not self.queue.empty()):
            tempCost, tempNode, tempPath = self.queue.get()

            tempCost -= self.heuristicMap[tempNode][goalNode]

            if (tempNode == goalNode):
                self.found = True
            else:
                self.awakenNode(tempCost, tempNode, tempPath, goalNode)

        # Put the result to attributes
        self.resultPath = tempPath + [tempNode]
        self.distance = tempCost

        # Clear the queue and visited set
        while (not self.queue.empty()):
            self.queue.get()
        self.visited.clear()
```

Gambar 3.1. *Source code file* Astar.py

3.2 *File* UCS.py

```python
# File : UCS.py
# Contains the class UCS, that are used for
# Finding the shortest path on a graph using the UCS algorithm

from queue import PriorityQueue

class UCS:

    # Constructor
    def __init__(self, adjacency_matrix, start_node, node_name):
        self.adjacency_matrix = adjacency_matrix
        self.start_node = start_node
        self.node_name = node_name
        self.path_result = [] # Notes: 0 means the 1st node, 1 means the 2nd node, etc.
        self.distance = 0
        self.foundPath = False

    # Getter
    def getAdjacencyMatrix(self):
        return self.adjacency_matrix

    def getStartNode(self):
        return self.start_node

    def getPathResult(self):
        return self.path_result

    def getDistance(self):
        return self.distance

    def getFoundPath(self):
        return self.foundPath

    # Path Result in Alphabets
    # Can only be used if foundPath = True
    def printPathResult(self):
        path_res = ""
        for i in range(0, len(self.path_result)-1):
            path_res += self.node_name[self.path_result[i]] + " - "
        path_res += self.node_name[self.path_result[len(self.path_result)-1]]
        return path_res
```

```python
    # Uniform Cost Search (UCS) Algorithm
    # Find shortest path and distance from start node to goal node
    def find_path_UCS(self, goal_node):
        i = 0
        visited_nodes = set()
        node_queue = PriorityQueue()
        node_queue.put((0, self.start_node, []))

        # There are still nodes to explore
        while (not node_queue.empty()):
            total_weight, current_node, path = node_queue.get()
            # Reach goal node
            if (current_node == goal_node):
                self.path_result = path + [current_node]
                self.distance = total_weight
                self.foundPath = True
                break

            # If the current node is not visited before, add to visited nodes
            if (not (current_node in visited_nodes)):
                visited_nodes.add(current_node)
                for neighbor_node, edge_weight in enumerate(self.adjacency_matrix[current_node]):
                    if (neighbor_node not in visited_nodes and edge_weight > 0):
                        new_path = path + [current_node]
                        new_distance = total_weight + edge_weight
                        node_queue.put((new_distance, neighbor_node, new_path))

        if (not self.foundPath):
            print("Lintasan tidak ditemukan")
```

Gambar 3.2. *Source code file* UCS.py

3.3 *File* Parser.py

```python
class Parser:

    # CONSTRUCTOR
    # Construct and initialize the object
    def __init__(self, fileName):
        self.row = -1
        self.col = -1
        self.map = [[]]
        self.heuristicMap = [[]]
        self.nodeName = []
        self.fileName = fileName

    # GETTER
    # Get the row of matrix
    def getRow(self):
        return self.row

    # Get the column of matrix
    def getCol(self):
        return self.col

    # Get the adjacency matrix
    def getMap(self):
        return self.map

    # Get the heuristic matrix
    def getHeuristicMap(self):
        return self.heuristicMap

    # Get the list of node name
    def getNodeName(self):
        return self.nodeName
```

```python
    # HELPER METHOD
    # Check if the map and heuristic map symmetric or not
    def isMapSymmetric(self):
        i = 0
        isSymmetric = True

        while (i < self.row and isSymmetric):
            j = i+1
            while (j < self.col and isSymmetric):
                if (self.map[i][j] != self.map[j][i] or self.heuristicMap[i][j] != self.heuristicMap[j][i]):
                    isSymmetric = False
                else:
                    j += 1
            if (isSymmetric):
                i += 1

        return isSymmetric

    # PARSING METHOD
    # Parse the line and fill it to nodeName attribute
    def parseNodeName(self, line):
        nameList = line.split(" ")

        # Check if there is less than 8 nodes
        if (len(nameList) < 8):
            raise Exception("Please input a map/graph that contain 8 nodes or more")

        self.row = len(nameList)
        self.col = len(nameList)

        # Put all the name of node
        for name in nameList:
            self.nodeName.append(name.strip("\n"))
```

```
# Parse every line on lines and fill it to map attribute
def parseMap(self, lines):
    self.map = [[0 for j in range(self.col)] for i in range(self.row)]

    # Put all elements to map
    for i in range (0, self.row):
        line = lines[i].split(" ")

        # Check if the column for this line is the same as number of node
        if (len(line) != self.col):
            raise Exception("Different number of columns in a map row")

        for j in range(0, self.col):
            # Check if the element does not follow the format
            if (i != j and float(line[j]) < 0):
                raise Exception("The weight of the sides of map/graph cannot be < 0")
            if (i == j and float(line[j]) != 0):
                raise Exception("The map main diagonal elements must be 0")

            self.map[i][j] = float(line[j])

# Parse every line on lines and fill it to map attribute
def parseHeuristic(self, lines):
    self.heuristicMap = [[0 for j in range(self.col)] for i in range(self.row)]

    # Put all elements to map
    for i in range (0, self.row):
        line = lines[i].split(" ")

        # Check if the column for this line is the same as number of node
        if (len(line) != self.col):
            raise Exception("Different number of columns in a heuristic map row")

        for j in range(0, self.col):
            # Check if the element does not follow the format
            if (i != j and float(line[j]) <= 0):
                raise Exception("The straight line distance of two node cannot \u2264 0")
            if (i == j and float(line[j]) != 0):
                raise Exception("The heuristic map main diagonal elements must be 0")

            self.heuristicMap[i][j] = float(line[j])
```

```
# Open the map/graph file and start the parse
def openAndParse(self):
    # Check if the file is in txt format or not
    if (not self.fileName.endswith(".txt")):
        raise Exception("Invalid file format, must be txt")

    # Open and read all lines of file
    with open(f"{self.fileName}", "r") as file:
        lines = file.readlines()
        file.close()

    # Check if the file is empty
    if (len(lines) == 0):
        raise Exception('The file is empty')

    # Parse the first lines, get all of node names
    self.parseNodeName(lines[0])

    # Check if there is enough remaining lines for map
    if (len(lines)-1 < self.row):
        raise Exception("There is too few row for map")

    # Parse the map, get the adjacency matrix
    self.parseMap(lines[1:1+self.row])

    # Check if there is exactly enough remaining lines for heuristic map
    if (len(lines)-1-self.row != self.row):
        raise Exception("There is too few or too much row for heuristic map")

    # Parse the heuristic map
    self.parseHeuristic(lines[1+self.row:1+self.row+self.row])

    # Check if map and heuristic map symmetric or not
    if (not self.isMapSymmetric() ):
        raise Exception("One of the sides in map/graph has two different weight")
```

Gambar 3.3. *Source code file* Parser.py

3.4 *File* GraphDrawer.py

```python
# File : GraphDrawer.py
# Contains the class GraphDrawer, that are used for
# Draw a graph or a graph with result path and save it as a file,

import matplotlib.pyplot as plt
import networkx as nx

class GraphDrawer:

    # CONSTRUCTOR
    # Construct and initialize the object
    def __init__(self, nodeName, row, map):
        self.nodeName = nodeName
        self.graph = nx.Graph()
        # Add the node to graph
        for i in range (len(self.nodeName)):
            self.graph.add_node(f"{i+1} - {self.nodeName[i]}")

        # Add the edge and its weight to graph
        for i in range(row):
            for j in range(i, row):
                if (map[i][j] != 0):
                    self.graph.add_edge(f"{i+1} - {self.nodeName[i]}", f"{j+1} - {self.nodeName[j]}", weight=map[i][j])
                    self.graph.add_edge(f"{j+1} - {self.nodeName[j]}", f"{i+1} - {self.nodeName[i]}", weight=map[j][i])

    # TRANSFORMER
    # Transform the result path into list of edges needed
    def transformResultPath(self, resultPath):
        edgeList = []
        for i in range(len(resultPath)-1):
            edgeList.append((f"{resultPath[i] + 1} - {self.nodeName[resultPath[i]]}", f"{resultPath[i+1] + 1} - {self.nodeName[resultPath[i+1]]}"))
            edgeList.append((f"{resultPath[i+1] + 1} - {self.nodeName[resultPath[i+1]]}", f"{resultPath[i] + 1} - {self.nodeName[resultPath[i]]}"))
        return edgeList

    # DRAWER
    # Draw a graph based on its nodes, edges, and node's name
    def drawGraph(self):
        # Create the image of graph and save it
        edges = [(u, v) for (u, v, d) in self.graph.edges(data=True)]
        edge_labels = nx.get_edge_attributes(self.graph, "weight")
        try:
            pos = nx.planar_layout(self.graph)
        except:
            pos = nx.spring_layout(self.graph, seed=len(self.nodeName))

        nx.draw_networkx_nodes(self.graph, pos, node_size=350, node_color="#EE1AEF")
        nx.draw_networkx_labels(self.graph, pos, font_size=12, font_family="monospace")
        nx.draw_networkx_edges(self.graph, pos, edgelist=edges, width=2)
        if (len(self.nodeName) < 15):
            nx.draw_networkx_edge_labels(self.graph, pos, edge_labels, rotate=False)

        ax = plt.gca()
        ax.margins(0.08)
        plt.axis("off")
        plt.tight_layout()
        plt.savefig("../assets/graph.png")
        plt.close("all")

    # Draw a graph based on its nodes, edges, node's name, and highlight the result path
    def drawGraphResult(self, resultPath):
        # Get all the result edge
        resultEdge = self.transformResultPath(resultPath)

        # Create the image of graph and save it
        normalEdges = [(u, v) for (u, v, d) in self.graph.edges(data=True) if (u, v) not in resultEdge]
        resultEdges = [(u, v) for (u, v, d) in self.graph.edges(data=True) if (u, v) in resultEdge]
        edge_labels = nx.get_edge_attributes(self.graph, "weight")
        try:
            pos = nx.planar_layout(self.graph)
        except:
            pos = nx.spring_layout(self.graph, seed=len(self.nodeName))

        nx.draw_networkx_nodes(self.graph, pos, node_size=350, node_color="#EE1AEF")
        nx.draw_networkx_labels(self.graph, pos, font_size=12, font_family="monospace")
        nx.draw_networkx_edges(self.graph, pos, edgelist=normalEdges, width=2, edge_color="#000000")
        nx.draw_networkx_edges(self.graph, pos, edgelist=resultEdges, width=2, edge_color="#FF1100")
        if (len(self.nodeName) < 15):
            nx.draw_networkx_edge_labels(self.graph, pos, edge_labels, rotate=False)

        ax = plt.gca()
        ax.margins(0.08)
        plt.axis("off")
        plt.tight_layout()
        plt.savefig("../assets/graph.png")
        plt.close("all")
```

Gambar 3.4. *Source code file* GraphDrawer.py

3.5 *File* main.py

```python
# File : main.py
# Contains the necessary code to make GUI and implement A* and UCS Algorithm

from tkinter import *
from tkinter.filedialog import askopenfilename
from Helper.Parser import *
from Helper.GraphDrawer import *
from Algorithm.UCS import *
from Algorithm.AStar import *
from PIL import Image, ImageTk

"""======================================== INITIALIZE SEGMENT ========================================"""
# Root Window
root = Tk()
root.title("PathFinder - Using UCS and A*")
root.resizable(0,0)

# Center the Window
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
window_width = 1080
window_height = 608
x = (screen_width // 2) - (window_width // 2)
y = (screen_height // 2) - (window_height // 2)
root.geometry(f"{window_width}x{window_height}+{x}+{y - 30}")

# Canvas
background = PhotoImage(file="../assets/background.png")
canvas = Canvas(root, width = 1080, height=608)
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, image = background, anchor = "nw")

# Basic UI Label and Shape
canvas.create_text(400, 10, anchor="nw", text= "\u2605 Path Finder \u2605", font=("Segoe Script", 25, 'bold'), fill="#FFB3C1")
canvas.create_text(70, 160, anchor="nw", text= "Choose your graph file!", font=("Georgia", 16), fill="#FFFFFF")
canvas.create_text(70, 270, anchor="nw", text= "Choose the path to find!", font=("Georgia", 16), fill="#FFFFFF")
canvas.create_text(70, 300, anchor="nw", text= "Start Node: ", font=("Georgia", 16), fill="#FFFFFF")
canvas.create_text(70, 340, anchor="nw", text= "Goal Node: ", font=("Georgia", 16), fill="#FFFFFF")
canvas.create_text(70, 420, anchor="nw", text= "Choose the algorithm!", font=("Georgia", 16), fill="#FFFFFF")
canvas.create_rectangle(0, 70, 1440, 75, fill="#EE1AEF")
canvas.create_rectangle(0, 80, 1440, 85, fill="#EE1AEF")
canvas.create_rectangle(400, 85, 405, 1080, fill="#EE1AEF")
```

```python
"""======================================== FILE CHOOSING SEGMENT ========================================"""
# Global Variables needed
parser = Parser("")
drawer = GraphDrawer([], -1, [[]])
valid = False

# File Name Label
fileName = StringVar()
fileName.set("")
fileNameLabel = Label(canvas, textvariable=fileName, fg="#000000", bg="#FFB3C1", font=("Times New Roman", 16), borderwidth=2, anchor="w", height= 1, width=20, relief="ridge")
fileNameWindow = canvas.create_window(70, 195, anchor="nw", window=fileNameLabel)

# Parsing Exception Label
parseExceptionLabel = canvas.create_text(70, 240, text = "", font=("Times New Roman", 11), fill="#FFFFFF", anchor="w")

# Procedure when the choose file button is clicked
def askFileName():
    global valid, parser, drawer, fileName, canvas, parseExceptionLabel
    valid = False
    graphFile = askopenfilename()
    temp = graphFile.split("/")[len(graphFile.split("/")) - 1]
    if (len(temp) > 19):
        fileName.set(temp[:19] + "...")
    else:
        fileName.set(temp)
    if (graphFile != ""):
        try:
            parser = Parser(graphFile)
            parser.openAndParse()
            drawer = GraphDrawer(parser.getNodeName(), parser.getRow(), parser.getMap())
            updateOptionNode(parser.getRow())
            updateResult([], "", 0, 0)
            canvas.itemconfigure(parseExceptionLabel, text="")
            valid = True
        except ValueError:
            canvas.itemconfigure(parseExceptionLabel, text="Unknown symbol in the map/graph file")
        except Exception as error:
            canvas.itemconfigure(parseExceptionLabel, text=error)

# File Chooser Button
fileChooserButton = Button(canvas, text="\u2b60", font=("Times New Roman", 11), command=askFileName, height= 1, width=4, bg="#EE1AEF")
fileChooserWindow = canvas.create_window(275, 195, anchor="nw", window=fileChooserButton)
```

```python
"""======================================== NODE CHOOSING SEGMENT ========================================"""
# The Option Chosen
startChosen = StringVar()
goalChosen = StringVar()
node = [0]

# Start Node Option Menu
startNodeOption = OptionMenu(canvas, startChosen, *node)
startNodeOption.config(height=1, width=5, background="#FFB3C1")
startNodeWindow = canvas.create_window(190, 300, anchor="nw", window=startNodeOption)

# Goal Node Option Menu
goalNodeOption = OptionMenu(canvas, goalChosen, *node)
goalNodeOption.config(height=1, width=5, background="#FFB3C1")
goalNodeWindow = canvas.create_window(190, 340, anchor="nw", window=goalNodeOption)

# Update the nodes option according to the graph file
def updateOptionNode(row):
    global startChosen, goalChosen, startNodeOption, goalNodeOption
    node = [i+1 for i in range(row)]
    menu1 = startNodeOption["menu"]
    menu2 = goalNodeOption["menu"]
    menu1.delete(0, "end")
    menu2.delete(0, "end")
    for i in node:
        menu1.add_command(label=i, command= lambda value=i: startChosen.set(value))
        menu2.add_command(label=i, command= lambda value=i: goalChosen.set(value))
```

```python
"""===================================== PATH FINDING SEGMENT ======================================="""
# Running Algorithm Exception Label
algorithmExceptionLabel = canvas.create_text(70, 500, text = "", font=("Times New Roman", 11), fill="#FFFFFF", anchor="w")

# Check if the algorithm can be run
def validateInput():
    global valid, startChosen, goalChosen, canvas, algorithmExceptionLabel
    if (not valid):
        raise Exception("Please input a valid graph file")
    elif ((len(startChosen.get())==0 or int(startChosen.get()) == 0 or int(startChosen.get()) > parser.getRow()) and (len(startChosen.get())==0 or int(goalChosen.get()) == 0 or int(goalChosen.get()) > parser.getRow())):
        raise Exception("Please input a valid start/goal node")
    canvas.itemconfigure(algorithmExceptionLabel, text="")

# Start the UCS Algorithm
def findUCSPath():
    global parser, canvas, algorithmExceptionLabel
    try:
        validateInput()
        ucs = UCS(parser.getMap(), int(startChosen.get())-1, parser.getNodeName())
        ucs.find_path_UCS(int(goalChosen.get())-1)
        if (ucs.getFoundPath()):
            updateResult(ucs.getPathResult(), ucs.printPathResult(), ucs.getDistance(), 0)
        else:
            canvas.itemconfigure(algorithmExceptionLabel, text="The path does not exist")
    except Exception as error:
        canvas.itemconfigure(algorithmExceptionLabel, text=error)

# UCS Button
ucsButton = Button(canvas, text="UCS", font=("Times New Roman", 15, 'bold'), command=findUCSPath, height=1, width=8, bg="#EE1AEF")
ucsWindow = canvas.create_window(70, 450, anchor="nw", window=ucsButton)

# Start the A* Algorithm
def findAStarPath():
    global parser, canvas, algorithmExceptionLabel
    try:
        validateInput()
        aStar = AStar(parser.getNodeName(), parser.getMap(), parser.getHeuristicMap())
        aStar.findPath(int(startChosen.get())-1, int(goalChosen.get())-1)
        if (aStar.getFound()):
            updateResult(aStar.getResultPath() ,aStar.getPathName(), aStar.getDistance(), 1)
        else:
            canvas.itemconfigure(algorithmExceptionLabel, text="The path does not exist")
    except Exception as error:
        canvas.itemconfigure(algorithmExceptionLabel, text=error)

# A* Button
aStarButton = Button(canvas, text="A*", font=("Times New Roman", 15, 'bold'), command=findAStarPath, height=1, width=8, bg="#EE1AEF")
aStarWindow = canvas.create_window(205, 450, anchor="nw", window=aStarButton)
```

```python
"""===================================== SHOWING RESULT SEGMENT ======================================="""
# Graph Image
graphImage = PhotoImage(file="../assets/logo.png")
graphWindow = canvas.create_image(500, 125, anchor="nw", image=graphImage)

# Algorithm Used Label
algorithmUsedLabel = canvas.create_text(500, 510, text = "", font=("Georgia", 12), fill="#FFFFFF", anchor="w")

# Result Path Label
resultPathLabel = canvas.create_text(500, 540, text = "", font=("Georgia", 12), fill="#FFFFFF", anchor="w")

# Distance Label
distanceLabel = canvas.create_text(500, 570, text = "", font=("Georgia", 12), fill="#FFFFFF", anchor="w")

# Update result according to parameter
def updateResult(resultPath, pathName, distance, flag):
    global drawer, canvas, graphImage, graphWindow, resultPathLabel, distanceLabel
    if (resultPath != []):
        drawer.drawGraphResult(resultPath)
        if (flag == 0):
            canvas.itemconfigure(algorithmUsedLabel, text="Using UCS Algorithm")
        else:
            canvas.itemconfigure(algorithmUsedLabel, text="Using A* Algorithm")
        canvas.itemconfigure(resultPathLabel, text=f"Result Path: {pathName}")
        canvas.itemconfigure(distanceLabel, text=f"Distance      : {distance}")
    else:
        drawer.drawGraph()
        canvas.itemconfigure(algorithmUsedLabel, text="")
        canvas.itemconfigure(resultPathLabel, text="")
        canvas.itemconfigure(distanceLabel, text="")

    # Open the image file
    image = Image.open("../assets/graph.png")

    # Calculate the new size
    new_width = int(image.width * 0.75)
    new_height = int(image.height * 0.75)

    # Resize the image
    resized_image = image.resize((new_width, new_height), Image.LANCZOS)
    graphImage = ImageTk.PhotoImage(resized_image)
    canvas.itemconfigure(graphWindow, image=graphImage)


root.mainloop()
```

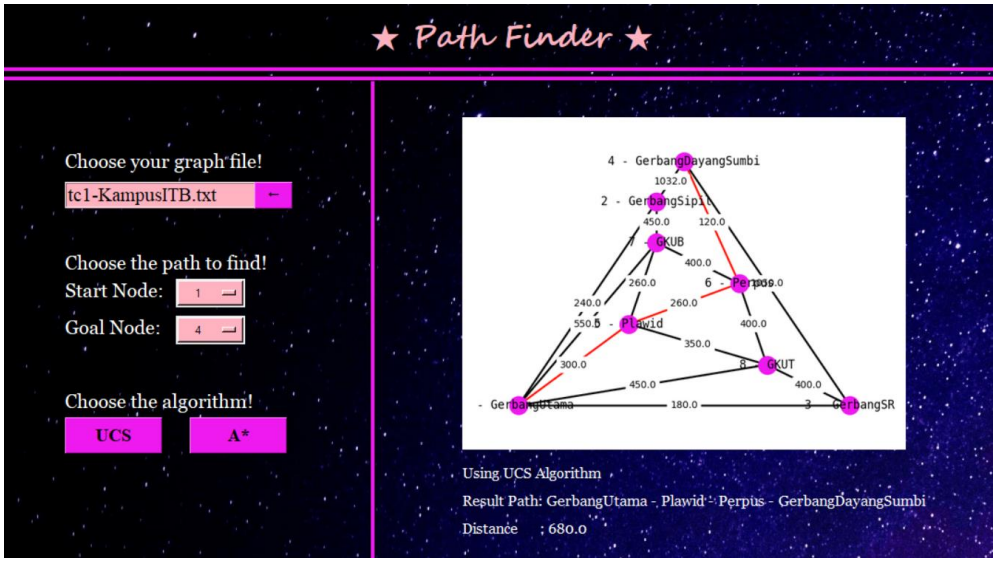
Gambar 3.5. *Source code file* main.py

Bab 4 : Pengujian Program

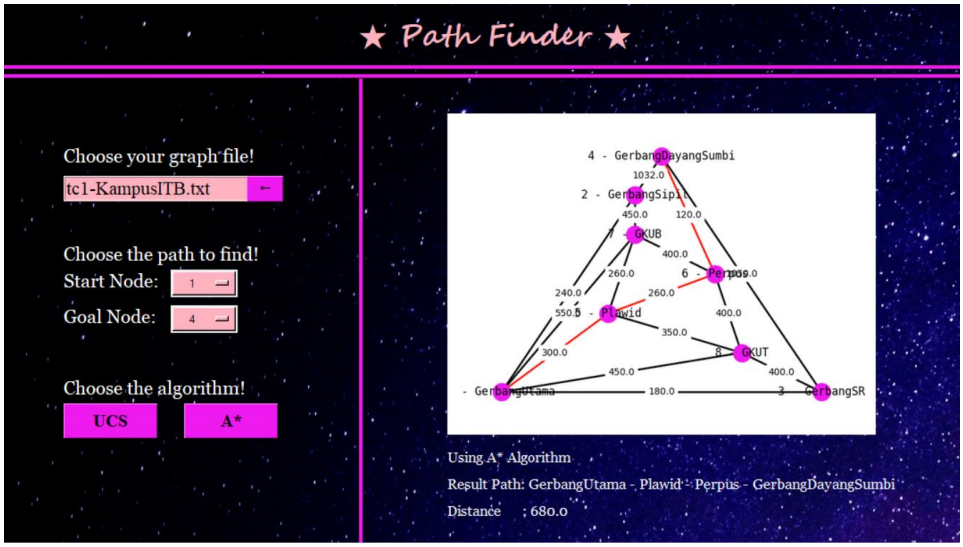
4.1 Pengujian *testcase* 1:

```
GerbangUtama GerbangSipil GerbangSR GerbangDayangSumbi Plawid Perpus GKUB GKUT
0 240 180 0 300 0 550 450
240 0 0 1032 0 0 450 0
180 0 0 1050 0 0 0 400
0 1032 1050 0 0 120 0 0
300 0 0 0 0 260 260 350
0 0 0 120 260 0 400 400
550 450 0 0 260 400 0 0
450 0 400 0 350 400 0 0
0 232.02 172.12 654.13 308.67 549.58 350.38 359.17
232.02 0 388.96 781.15 432.72 656.58 379.57 549.90
172.12 388.96 0 691.36 397.35 616.75 473.78 353.51
654.13 781.15 691.36 0 354.32 143.06 427.11 341.02
308.67 432.72 397.35 354.32 0 238.86 142.85 181.40
549.58 656.58 616.75 143.06 238.86 0 293.68 300.17
350.38 379.57 473.78 427.11 142.85 293.68 0 313.64
359.17 549.90 353.51 341.02 181.40 300.17 313.64 0
```

Gambar 4.1.1 Masukan *file testcase* 1 (Peta Kampus ITB)



Gambar 4.1.2 Pencarian Lintasan Terpendek dengan Algoritma UCS

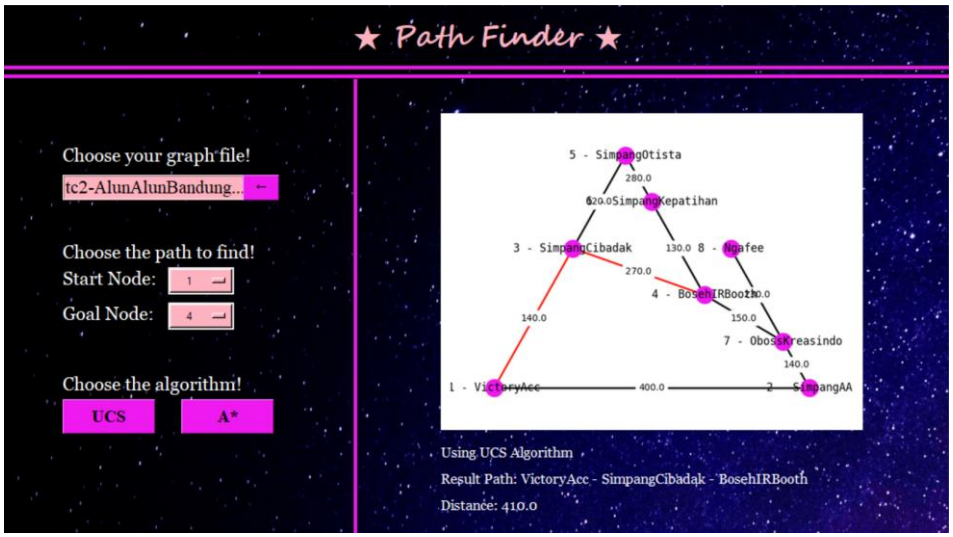


Gambar 4.1.3 Pencarian Lintasan Terpendek dengan Algoritma A*

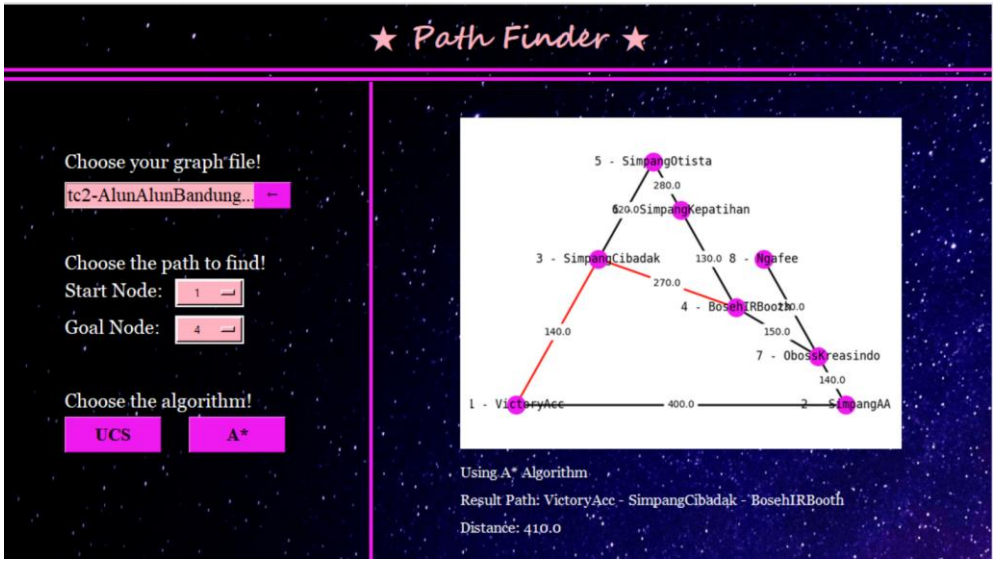
4.2 Pengujian *testcase* 2:

```
VictoryAcc SimpangAA SimpangCibadak BosehIRBooth SimpangOtista SimpangKepatihan ObossKreasindo Ngafee
0 400 140 0 0 0 0 0
400 0 0 0 0 0 140 0
140 0 0 270 120 0 0 0
0 0 270 0 0 130 150 0
0 0 120 0 0 280 0 0
0 0 0 130 280 0 0 0
0 140 0 150 0 0 0 230
0 0 0 0 0 0 230 0
0 400 137 304 252 377 432 667
400 0 420 194 460 290 140 287
137 420 0 264 117 300 403 645
304 194 264 0 280 120 145 383
252 460 117 280 0 266 414 645
377 290 300 120 266 0 179 397
432 140 403 145 414 179 0 225
667 287 645 383 645 397 225 0
```

Gambar 4.2.1 Masukan file testcase 2 (Peta Alun-Alun Kota Bandung)



Gambar 4.2.2 Pencarian Lintasan Terpendek dengan Algoritma UCS

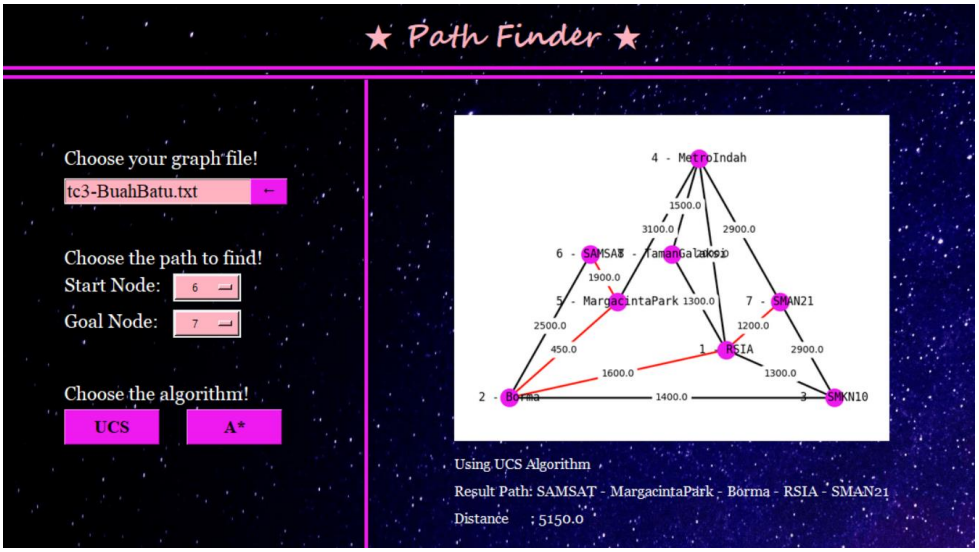


Gambar 4.2.3 Pencarian Lintasan Terpendek dengan Algoritma A*

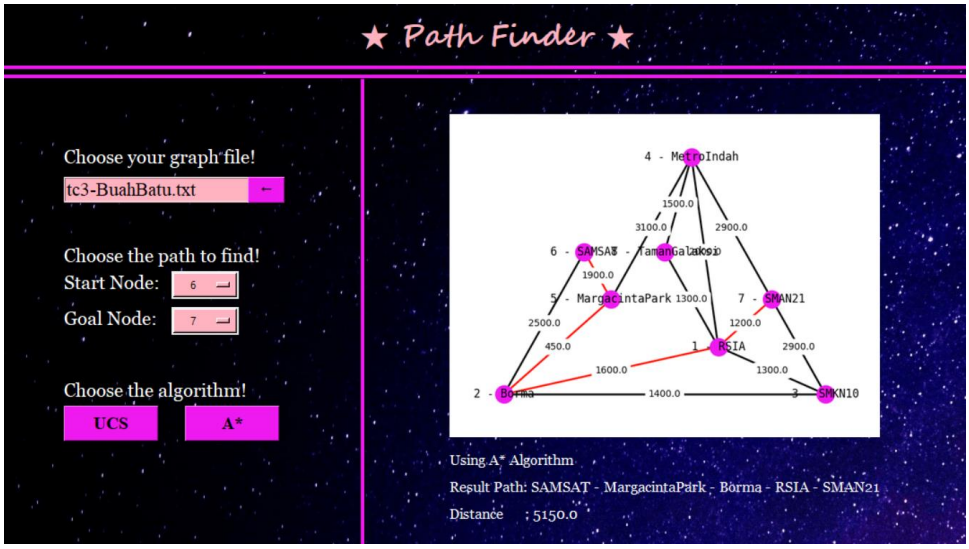
4.3 Pengujian *testcase* 3:

```
RSIA Borma SMKN10 MetroIndah MargacintaPark SAMSAT SMAN21 TamanGalaksi
0 1600 1300 2000 0 0 1200 1300
1600 0 1400 0 450 2500 0 0
1300 1400 0 0 0 0 2900 0
2000 0 0 0 3100 0 2900 1500
0 450 0 3100 0 1900 0 0
0 2500 0 0 1900 0 0 0
1200 0 2900 2900 0 0 0 0
1300 0 0 1500 0 0 0 0
0 1160 1040 1520 1580 2400 998.36 813.66
1160 0 1180 1640 420.65 1440 2150 666.86
1040 1180 0 2400 1480 2590 1690 1390
1520 1640 2400 0 1840 1780 2110 1050
1580 420.65 1480 1840 0 1140 2570 975.11
2400 1440 2590 1780 1140 0 3340 1580
998.36 2150 1690 2110 2570 3340 0 1750
813.66 666.86 1390 1050 975.11 1580 1750 0
```

Gambar 4.3.1 Masukan *file testcase* 3 (Peta Kecamatan Buah Batu)



Gambar 4.3.2 Pencarian Lintasan Terpendek dengan Algoritma UCS

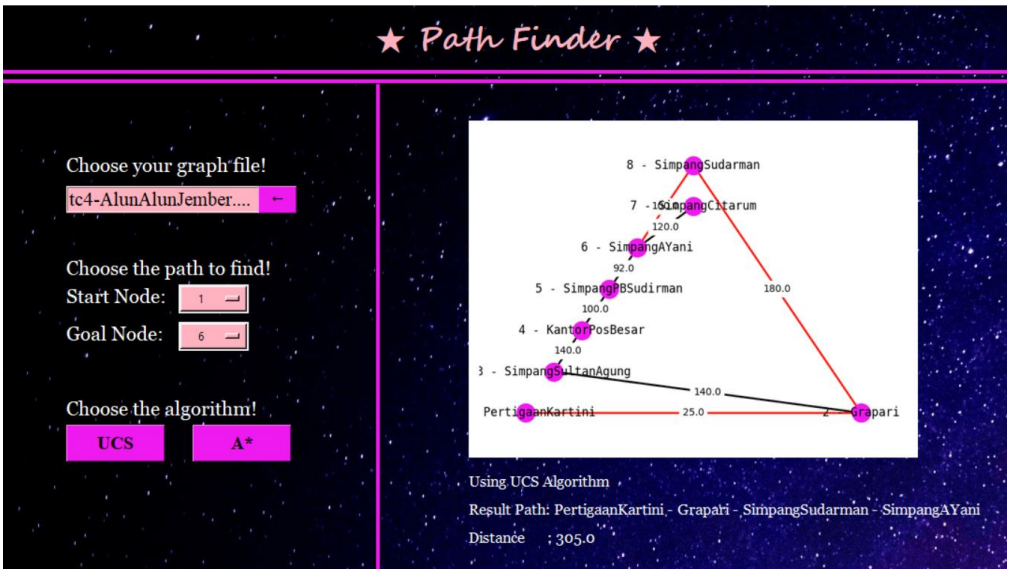


Gambar 4.3.3 Pencarian Lintasan Terpendek dengan Algoritma A*

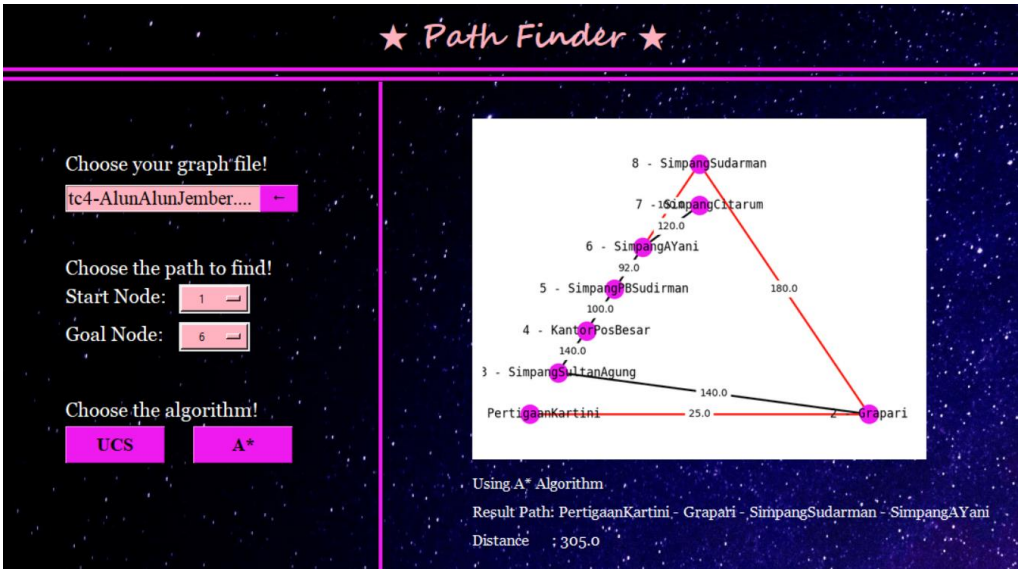
4.4 Pengujian *testcase* 4:

```
PertigaanKartini Grapari SimpangSultanAgung KantorPosBesar SimpangPBSudirman SimpangAYani SimpangCitarum SimpangSudarman
0 25 0 0 0 0 0 0
25 0 140 0 0 0 0 180
0 140 0 140 0 0 0 0
0 0 140 0 100 0 0 0
0 0 0 100 0 92 0 0
0 0 0 0 92 0 120 100
0 0 0 0 0 120 0 0
0 180 0 0 0 100 0 0
0 37 154 213 267 238 321 178
37 0 112 183 234 212 304 174
154 112 0 79 166 181 292 209
213 183 79 0 99 136 251 214
267 234 166 99 0 75 162 179
238 212 181 136 75 0 109 111
321 304 292 251 162 109 0 153
178 174 209 214 179 111 153 0
```

Gambar 4.4.1 Masukan *file testcase* 4 (Peta Alun-Alun Kabupaten Jember)



Gambar 4.4.2 Pencarian Lintasan Terpendek dengan Algoritma UCS

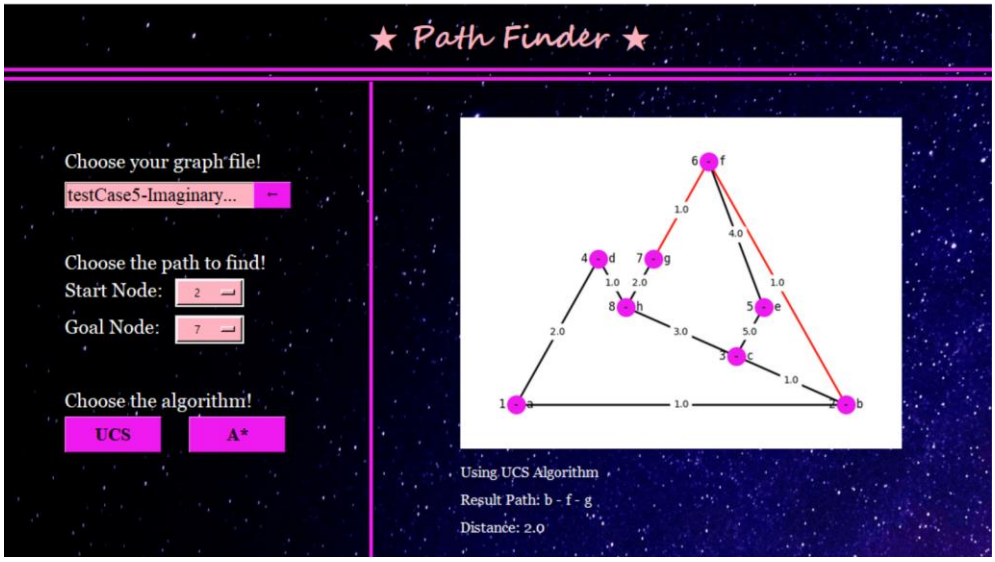


Gambar 4.4.3 Pencarian Lintasan Terpendek dengan Algoritma A*

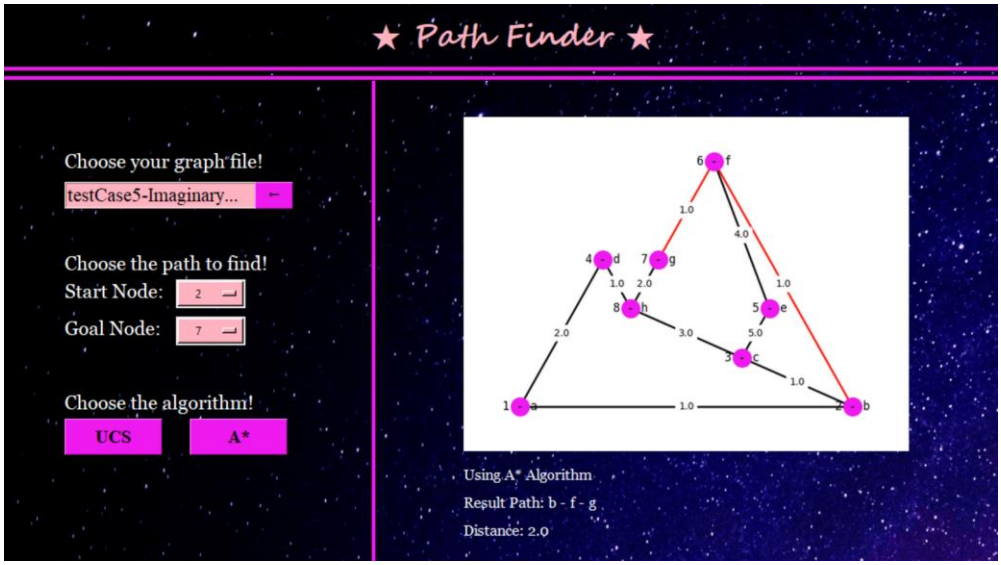
4.5 Pengujian *testcase* 5:

```
a b c d e f g h
0 1 0 2 0 0 0 0
1 0 1 0 0 1 0 0
0 1 0 0 5 0 0 3
2 0 0 0 0 0 0 1
0 0 5 0 0 4 0 0
0 1 0 0 4 0 1 0
0 0 0 0 0 1 0 2
0 0 3 1 0 0 2 0
0 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1
1 1 0 1 1 1 1 1
1 1 1 0 1 1 1 1
1 1 1 1 0 1 1 1
1 1 1 1 1 0 1 1
1 1 1 1 1 1 0 1
1 1 1 1 1 1 1 0
```

Gambar 4.5.1 Masukan file testcase 5 (Peta Khayalan)



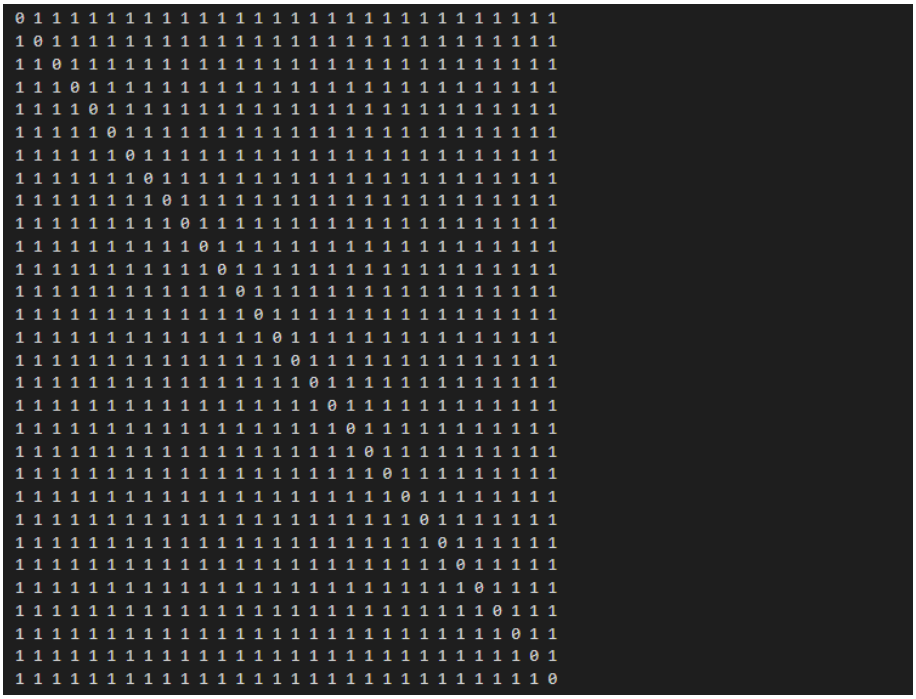
Gambar 4.5.2 Pencarian Lintasan Terpendek dengan Algoritma UCS



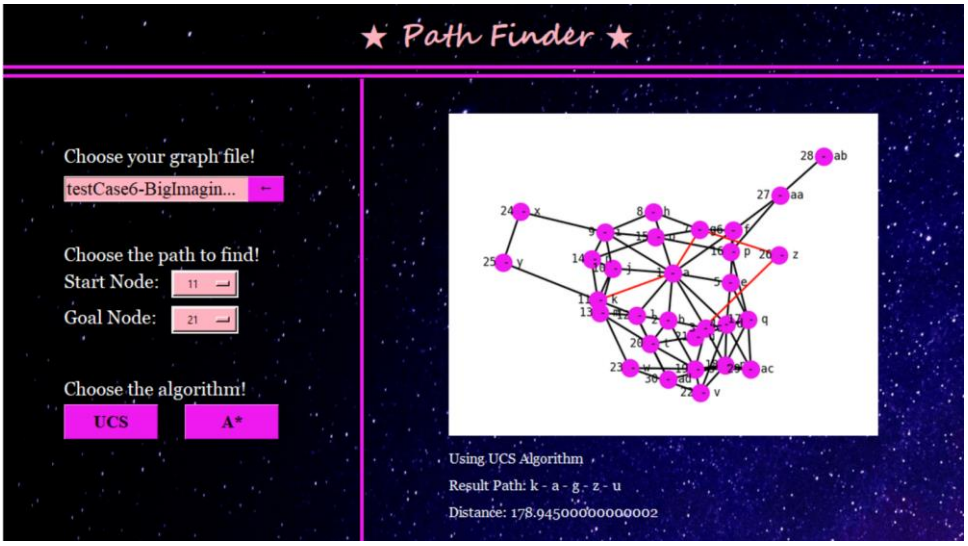
Gambar 4.5.3 Pencarian Lintasan Terpendek dengan Algoritma A*

4.6 Pengujian *testcase* 6:

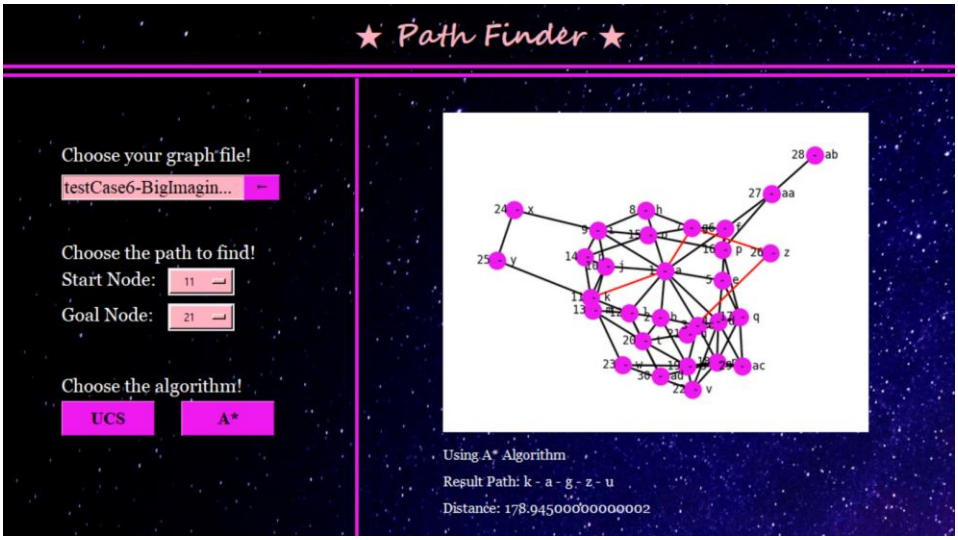
```
a b c d e f g h i j k l m n o p q r s t u v w x y z aa ab ac ad
0 61.099 48.516 62.234 73.12 76.305 70.709 58.249 54.606 60.708 62.069 78.906 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
61.099 0 45.392 0 0 0 0 0 0 0 28.138 0 0 0 0 0 0 44.835 52.132 0 0 0 0 0 0 0 0 0
48.516 45.392 0 52.879 0 0 0 0 0 0 0 0 0 0 0 0 0 41.321 46.52 0 0 0 0 0 0 0 0 0
62.234 0 52.879 0 35.717 0 0 0 0 0 0 0 0 0 0 0 56.67 37.295 0 0 0 33.867 0 0 0 0 0 0 36.015 0
73.12 0 0 35.717 0 23.618 0 0 0 0 0 0 0 0 0 0 43.278 0 0 0 0 0 0 0 0 0 0 0 0 0
76.305 0 0 0 23.618 0 18.698 0 0 0 0 0 0 0 0 0 37.281 0 0 0 0 0 0 0 0 0 0 57.07 0 0 0
70.709 0 0 0 0 18.698 0 28.654 0 0 0 0 0 0 0 47.492 0 0 0 0 0 0 0 0 0 0 0 30.891 0 0 0 0
58.249 0 0 0 0 0 28.654 0 31.788 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
54.606 0 0 0 0 0 0 31.788 0 60.412 0 0 0 39.535 45.442 0 0 0 0 0 0 0 0 0 35.603 0 0 0 0 0 0
60.708 0 0 0 0 0 0 0 60.412 0 36.277 0 44.541 42.179 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
62.069 0 0 0 0 0 0 0 0 36.277 0 34.839 0 0 0 0 0 0 0 0 0 0 52.707 0 29.105 0 0 0 0 0
78.906 28.138 0 0 0 0 0 0 0 0 34.839 0 61.901 0 0 0 0 0 0 40.201 0 0 0 0 0 0 0 0 0 38.917
0 0 0 0 0 0 0 0 0 44.541 0 61.901 0 79.551 0 0 0 0 0 94.392 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 39.535 42.179 0 0 79.551 0 75.622 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 47.492 0 45.442 0 0 0 0 75.622 0 74.33 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 37.281 0 0 0 0 0 0 0 0 74.33 0 65.753 0 0 0 0 0 0 0 0 0 39.374 0 0 0
0 0 0 56.67 43.278 0 0 0 0 0 0 0 0 0 0 65.753 0 80.997 0 0 65.54 0 0 0 0 0 0 0 45.569 0
0 0 41.321 37.295 0 0 0 0 0 0 0 0 0 0 0 80.997 0 67.543 0 0 46.377 0 0 0 0 0 0 38.953 89.742
0 44.835 46.52 0 0 0 0 0 0 0 0 0 0 0 0 0 0 67.543 0 60.495 0 75.598 59.122 0 0 0 0 0 106.453 0
0 52.132 0 0 0 0 0 0 0 0 0 40.201 94.392 0 0 0 0 0 60.495 0 146.772 0 0 0 0 0 0 0 0 34.97
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 65.54 0 0 146.772 0 0 0 0 0 15.276 0 0 0 0
0 0 0 33.867 0 0 0 0 0 0 0 0 0 0 0 0 0 0 46.377 75.598 0 0 0 35.307 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 52.707 0 0 0 0 0 0 0 59.122 0 0 35.307 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 35.603 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25.42 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 29.105 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25.42 0 0 0 0 0
0 0 0 0 0 0 30.891 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 15.276 0 0 0 0 0 0 0 0 0
0 0 0 0 0 57.07 0 0 0 0 0 0 0 0 0 39.374 0 0 0 0 0 0 0 0 0 0 0 0 70.475 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 70.475 0 0 0
0 0 0 36.015 0 0 0 0 0 0 0 0 0 0 0 0 0 45.569 38.953 106.453 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 38.917 0 0 0 0 0 89.742 0 34.97 0 0 0 0 0 0 0 0 0
```



Gambar 4.6.1 Masukan file testcase 6 (Peta Khayalan berukuran besar)



Gambar 4.6.2 Pencarian Lintasan Terpendek dengan Algoritma UCS

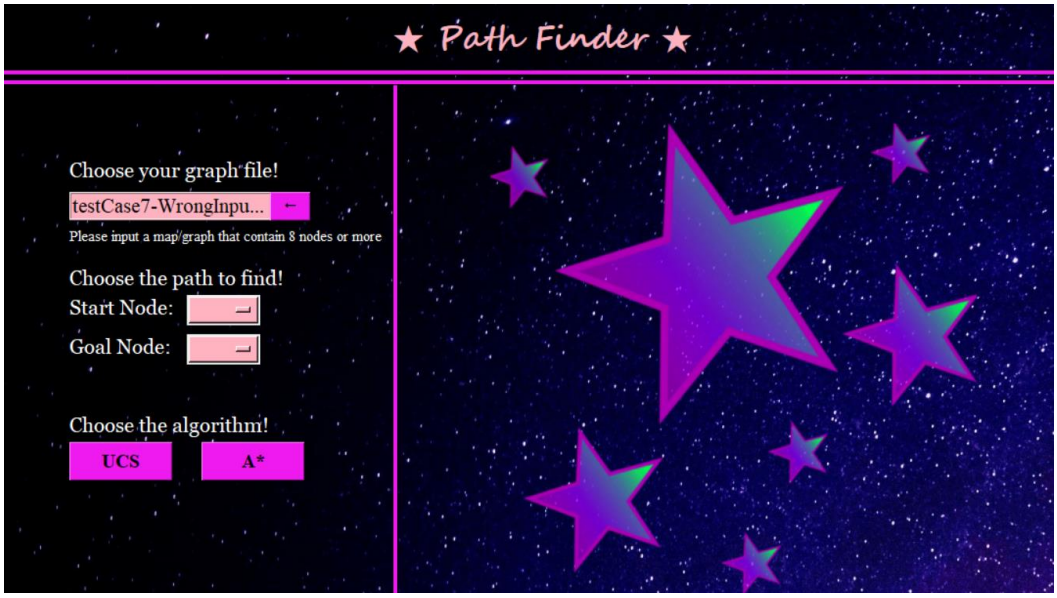


Gambar 4.6.3 Pencarian Lintasan Terpendek dengan Algoritma A*

4.7 Pengujian *testcase* 7:

```
a b c d
1 0 -1
3 2 4
1 0 9 9
-1 0 0
0 0 0
0 0 0
```

Gambar 4.7.1. Masukan *file testcase* 7 (Peta Input Salah)



Gambar 4.7.2. Output program (Masukan tidak valid)

Bab 5 : Kesimpulan

Dari pengerjaan tugas kecil III IF2211 Strategi Algoritma ini, kami memperoleh kesimpulan sebagai berikut.

1. Algoritma *Uniform Cost Search* (UCS) sebagai sebagai salah satu jenis *uninformed search* dan algoritma *A star* (A*) sebagai salah satu jenis *informed search* dapat digunakan untuk memperoleh jarak terpendek suatu lintasan dari suatu simpul ke simpul lain dalam suatu graf secara optimal.
2. Kami berhasil mengimplementasikan algoritma UCS dan A* untuk menemukan lintasan terpendek dari suatu simpul ke simpul yang lain dalam suatu graf dalam bahasa pemrograman Python.
3. Kami berhasil membuat GUI untuk memvisualisasikan hasil dari pencarian lintasan terpendek dari suatu simpul ke simpul yang lain dalam suatu graf dengan algoritma UCS dan A*.

Daftar Pustaka

Munir, R. (2021). Route Planning Bagian 1 [PDF file]. Retrieved April 8, 2023, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

Munir, R. (2021). Route Planning Bagian 2 [PDF file]. Retrieved April 8, 2023, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

Lampiran

Link Repository

Berikut adalah tautan *repository* untuk tugas kecil III IF2211 Strategi Algoritma ini.

Link : https://github.com/GoDillonAudris512/Tucil3_13521062_13521064

Tabel *Checklist*

No	Keterangan	Centang (✓) jika Iya
1	Program dapat menerima input graf.	✓
2	Program dapat menghitung lintasan terpendek dengan UCS.	✓
3	Program dapat menghitung lintasan terpendek dengan A*.	✓
4	Program dapat menampilkan lintasan terpendek serta jaraknya.	✓
5	Bonus: Program dapat menerima input peta dengan *Google Map API* dan menampilkan peta serta lintasan terpendek pada peta.	